EDGE++: Improved Training and Sampling of EDGE

Mingyang Wu*
Department of Computer Science
Tufts University
Medford, MA, USA
mingyang.wu@tufts.edu

Xiaohui Chen*
Department of Computer Science
Tufts University
Medford, MA, USA
xiaohui.chen@tufts.edu

Li-Ping Liu
Department of Computer Science
Tufts University
Medford, MA, USA
liping.liu@tufts.edu

Abstract

Recently developed deep neural models like NetGAN, CELL, and Variational Graph Autoencoders have made progress but face limitations in replicating key graph statistics on generating large graphs. Diffusion-based methods have emerged as promising alternatives, however, most of them present challenges in computational efficiency and generative performance. EDGE is effective at modeling large networks, but its current denoising approach can be inefficient, often leading to wasted computational resources and potential mismatches in its generation process. In this paper, we propose enhancements to the EDGE model to address these issues. Specifically, we introduce a degree-specific noise schedule that optimizes the number of active nodes at each timestep, significantly reducing memory consumption. Additionally, we present an improved sampling scheme that fine-tunes the generative process, allowing for better control over the similarity between the synthesized and the true network. Our experimental results demonstrate that the proposed modifications not only improve the efficiency but also enhance the accuracy of the generated graphs, offering a robust and scalable solution for graph generation tasks.

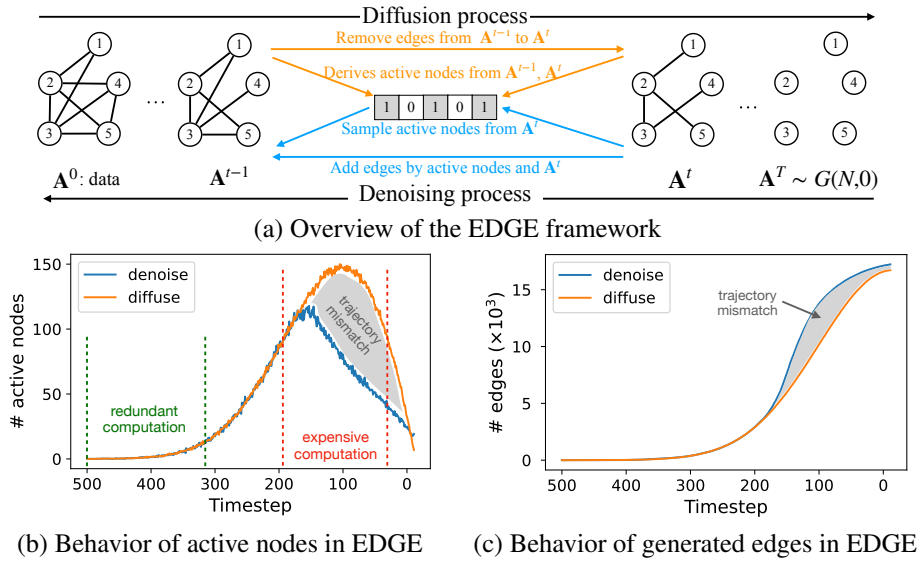
1 Introduction

The generation of large graphs has been accomplished using random graph models [24], such as the Stochastic-Block Model (SBM) [13]. Despite their use, these models fall short in capturing complex structures, paving the way to the development of deep neural models. Recently, several neural methods, including NetGAN [2], CELL [26], and Variational Graph Autoencoders [16], have been proposed to model large graphs. However, Chanpuriya et al. [3] points out that they are edge-independent model and are still incapable of reproducing key statistics unless they memorize the training graph, i.e., high edge overlap between the generated graphs and the original one. An edge-independent model generates all edges independently at once. In constrast, edge-dependent models like diffusion-based graph models [28; 12] and autoregressive graph models [31; 20], have shown promise with small graphs [14; 30]. In particular, diffusion-based graph models demonstrate better modeling capbility as they avoid the long-term memory issues.

While demonstrating significant success, the majority of existing diffusion models fail to generate large networks with thousands of nodes. Recently, Chen et al. [7] advocates to diffuse a graph into an empty graph and leverages a neural network to reverse the edge-removal process (see Fig. 1(a)). In the edge-removal process, it identifies that not all nodes participate in the edge-formation process

*These authors contributed equally to this work.

NeurIPS 2023 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2023).



(a) Overview of the EDGE framework

(b) Behavior of active nodes in EDGE (c) Behavior of generated edges in EDGE

Figure 1. (a) EDGE defines an edge-removal process and learns a model to reverse it. In the denoising process, it first identifies nodes that may have edges added (active node), then predicts edges between them. (b, c) Denoising and diffusion processes of active nodes and edges on Polblogs. The denoising process in EDGE is inconsistent with the reverse of the diffusion process. This mismatch issue appears in the active nodes prediction and consequently leads to unsatisfied generative performance.

in every timestep, and proposes to first select the active nodes then predict edge formation only among them. Furthermore, it shows that one can use a prescribed degree sequence to guide the graph generation. This denoising scheme notably reduces both the computational cost and task complexity.

While EDGE has demonstrated powerful capability in modeling large networks, the computational power of the denoising network is not fully utilized. As demonstrated in Fig. 1(b), by using default noise schedule in vanilla diffusion models, the computation is highly concentrated in the second half of the denoising process, wasting the network capability in the first half. Such an uneven distribution of the number of active nodes also leads to a higher memory consumption, as the space complexity is upperbounded by the largest number. Moreover, while using degree-guided generation, we observe that there is a mismatch between the diffusion and denoising processes (see Fig. 1(b,c)). Such inconsistency may potentially deteriorate the generative performance.

In this work, we propose two components to address the aforementioned issues in EDGE. First, we propose a degree-specific noise schedule to control the number of active nodes at each timestep. The proposed noise schedule significantly reduces the largest number of active nodes during denoising, saving much more computation in terms of memory. Second, we propose an improved sampling scheme, which fixes the generation error potentially made in each denoising step. Experiment results show that by adopting the proposed techniques, we obtain significant memory savings in training EDGE, and achieve better generative performance in Polblogs and PPI datasets. Moreover, we showcase how to perform graph generation with EO control by leveraging the proposed techniques.

2 Background

We are interested in a graph generative model that considers the following diffusion process over variables $\mathbf{A}^{1:T}$:

$$q(\mathbf{A}^{1:T}|\mathbf{A}^0) = \prod_{t=1}^{T} q(\mathbf{A}^t|\mathbf{A}^{t-1}), \text{ where } q(\mathbf{A}^t|\mathbf{A}^{t-1}) = \prod_{i,j:i<j} \mathcal{B}(\mathbf{A}^t_{i,j}; (1-\beta_t)\mathbf{A}^{t-1}_{i,j} + \beta_t p).$$

Here $\mathbf{A}^0$ is the data adjacency matrix, and $\mathcal{B}(x;\mu)$ is the Bernoulli distribution over variable x with probability μ. And β_t is the noise schedule parameter at timestep t. Following the convention, we define $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{\tau=1}^{t} \alpha_\tau$. Ideally, $\beta_{1:T}$ are tuned such that the marginal $q(\mathbf{A}^T|\mathbf{A}^0)$ converges to some prior distribution $p(\mathbf{A}^T)$. Specifically, $p(\mathbf{A}^T)$ is an Erdős-Rényi graph model $G(N,p)$, with N being the graph size and p being the probability an edge may occur between any two nodes. EDGE [7] set $p = 0$ and simplify the diffusion process into an edge-removal process

(Fig. 1(a)). Under this setting, it proposes two techniques to reduce the computation cost and learning complexity.

Introducing the active node variables. Due to the sparsity property of a graph, EDGE identifies that only a portion of the nodes may have their edges removed at every timestep, and derives the following equivalent diffusion process:

$$q(\mathbf{A}^{1:T}|\mathbf{A}^0) = q(\mathbf{A}^{1:T}, \mathbf{s}^{1:T}|\mathbf{A}^0) = \prod_{t=1}^{T} q(\mathbf{A}^t|\mathbf{A}^{t-1}) q(\mathbf{s}^t|\mathbf{A}^{t-1}, \mathbf{A}^t).$$

This is because $\mathbf{s}^t$ is deterministic given $\mathbf{A}^{t-1}$ and $\mathbf{A}^t$. In the denoising process, for each timestep, one can first decide which nodes will be active given $\mathbf{A}^t$, then only perform edge prediction among the active nodes. The denoising process can be formulated as

$$p_\theta(\mathbf{A}^{0:T}, \mathbf{s}^{1:T}) = p(\mathbf{A}^T) \prod_{t=1}^{T} p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t) p_\theta(\mathbf{s}^t|\mathbf{A}^t),$$

then the learning objective is to maximize the variational lower bound $\mathcal{L}(\mathbf{A}^0; \theta)$ of $\log p_\theta(\mathbf{A}^0)$:

$$\mathbb{E}_q \left[\log \frac{p(\mathbf{A}^T)}{q(\mathbf{A}^T|\mathbf{A}^0)} + \log p_\theta(\mathbf{A}^0|\mathbf{A}^1, \mathbf{s}^1) + \sum_{t=2}^{T} \log \frac{p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)}{q(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t, \mathbf{A}^0)} + \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{s}^t|\mathbf{A}^t)}{q(\mathbf{s}^t|\mathbf{A}^t, \mathbf{A}^0)} \right].$$

Degree-guided graph generation. EDGE further shows that if the initial degree $\mathbf{d}^0$ of the generated graph is given, one doesn't need to learn the active node predictor $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$. Specifically, it shows that given the initial degree $\mathbf{d}^0$ and the current degree $\mathbf{d}^t$, the active node posterior is

$$q(\mathbf{s}^t|\mathbf{d}^t, \mathbf{d}^0) = \prod_{i=1}^{N} q(\mathbf{s}_i^t|\mathbf{d}_i^t, \mathbf{d}_i^0), \text{ where } q(\mathbf{s}_i^t|\mathbf{d}_i^t, \mathbf{d}_i^0) = \mathcal{B}\left(\mathbf{s}_i^t; 1 - \left(1 - \frac{\beta_t \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\right)^{\mathbf{d}_i^0 - \mathbf{d}_i^t}\right). \quad (1)$$

To sample a graph, one needs to sample a degree sequence $\mathbf{d}^0$ first, then replace the parameterized active node distribution $p_\theta(\mathbf{s}^t|\mathbf{A}^t)$ with $q(\mathbf{s}^t|\mathbf{d}^t, \mathbf{d}^0)$. The degree-guided generation can significantly reduce the model learning complexity, greatly improving the generation accuracy.

Since EDGE defines latent variables on two levels of granularities – nodes and edges, the behavior of the active nodes should also be taken into consideration. Fig. 1(b) visualizes the node behavior when defining linear noise schedule on edges, the number of active nodes varies unevenly over timesteps, making the modeling of some timesteps wasteful and the sampling computation redundant. Moreover, we observe that there is a mismatch issue during sampling, making the generated graphs having a higher volume than the ground-truth graph (Fig 1(c)). Next, we present two techniques to address those limitations without modifying the framework of EDGE.

3 Methodologies

In this section, we first elaborate on how to improve the noise schedule based on active node control, and then we present volume-preserved sampling, which alleviates the aforementioned mismatching issue.

3.1 Improved noise schedule

While EDGE uses existing noise schedule schemes, which are defined on edges, we argue that a better schedule principle should focus on the nodes. Denote $\gamma_{1:T}$ to be an active node schedule, where $\gamma_t > 0$ for all t is an unnormalized portion of the total number of nodes. The reason it is unnormalized is that the actual number of active nodes is data-specific. We introduce a free parameter K such that $g_{\gamma_t}(K) := KN\gamma_t$ is the actual number of active nodes, we defer the discussion of the use of K later in this section.

Given $\gamma_{1:T}$, the goal is to find the corresponding edge noise schedule $\alpha_{1:T}$. In the following, we will first draw the connection between the edge noise schedule and the expected number of active nodes for each timestep, then we show how to obtain the parameters $\alpha_{1:T}$ that satisfy the given active node schedule.

Connecting edge noise schedule to active node control. Since the active node schedule is data-specific, let $\mathbf{d}^0 \in \mathbb{N}^N$ be the degree sequence of a graph, given $\alpha_{1:T}$, the expected number of active nodes $h_{\mathbf{d}^0}(\alpha_{1:t}, t) \in \mathbb{R}$ at timestep t is

$$h_{\mathbf{d}^0}(\alpha_{1:t}, t) = \begin{cases} \sum_{i=1}^{N} (1 - \alpha_1^{\mathbf{d}_i^0}), & \text{if } t = 1 \\ \sum_{i=1}^{N} \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} (1 - \alpha_t^{\mathbf{d}_i^{t-1}}) \operatorname{Bin}(k = \mathbf{d}_i^{t-1}, n = \mathbf{d}_i^0, p = \bar{\alpha}_{t-1}), & \text{otherwise} \end{cases}. \tag{2}$$

Here $\operatorname{Bin}(k, n, p)$ is a binomial distribution parameterized by number of trails n and probability p. Intuitively, the expected number of active nodes is computed as $\sum_{i=1}^{N} q(\mathbf{s}_i^t = 1|\mathbf{d}_i^0)$, where $q(\mathbf{s}_i^t|\mathbf{d}_i^0)$ is the distribution of active node i at timestep t. When $t > 1$, $q(\mathbf{s}_i^t|\mathbf{d}_i^0)$ is computed by the marginalization

$$q(\mathbf{s}_i^t|\mathbf{d}_i^0) = \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} q(\mathbf{s}_i^t|\mathbf{d}_i^{t-1}) q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^0), \tag{3}$$

where both $q(\mathbf{s}_i^t|\mathbf{d}_i^{t-1})$ and $q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^0)$ can be expressed analytically [7]. We further provide a detailed derivation in the App. A.

Finding the corresponding edge noise schedule $\alpha^*_{1:T}$. With the shown relation between $\alpha_{1:T}$ and active node behavior, we now can get $\alpha^*_{1:T}$ for any active node schedule. Specifically, we can obtain $\alpha^*_{1:T}$ by solving

$$\alpha^*_{1:T} = \arg\min_{\alpha_{1:T}} \underbrace{\sum_{t=1}^{T} \left(h_{\mathbf{d}^0}(\alpha_{1:t}, t) - g_{\gamma_t}(K) \right)^2}_{\mathcal{L}(\alpha_{1:T}; K, \gamma_{1:T}, \mathbf{d}^0)}, \text{ s.t. } \prod_{t=1}^{T} \alpha_t \approx 0. \tag{4}$$

Here we are actually optimizing $\alpha_{1:T}$ such that the expected number of active nodes matches the desired one (i.e., $g_{\gamma_t}(K)$) at each timestep. Recall that we need the marginal distribution $q(\mathbf{A}^T|\mathbf{A}^0)$ converge to $G(N, 0)$, so $\prod \alpha_t$ should converge to 0. This constraint is imposed when solving the objective.

To solve for $\alpha^*_{1:T}$ that satisfies the constraint, we introduce the parameter K, which is solved along with $\alpha_{1:T}$. The parameter K is tuned such that (1) the objective loss is sufficiently low (when K is small enough); (2) and the constraint $\prod \alpha_t \approx 0$ is satisfied (when K is large enough). In practice, we solve K and $\alpha^*_{1:T}$ alternatively via binary search (See Alg. 1). Given specific K, we solve the objective using a numerical solver.

3.2 Volume-preserved Sampling

We identify that in EDGE, the use of degree-guided posterior $q(\mathbf{s}|\mathbf{d}^t, \mathbf{d}^0)$ may lead to inconsistency between the active node behavior in diffusion and denoising processes. The reason is that when sampling edges from edge model $p_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$, the degree constraints are not imposed. As a result, some nodes may stay inactive with more edges than their degree specification, while other nodes that are under the budget will still be sampled to be active. Such error can not be fixed by the degree-guided posterior and thus will accumulate over the denoising process.

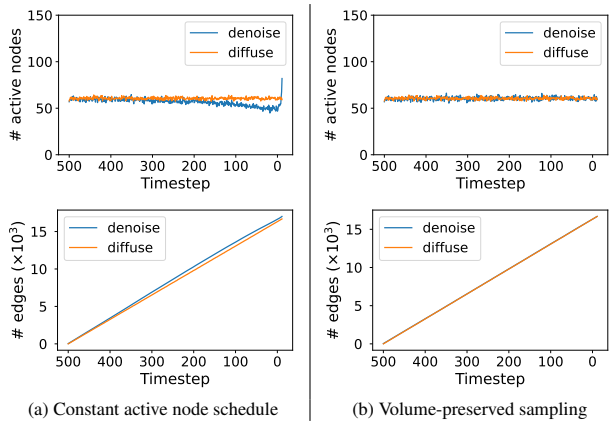


Figure 2. (a) Customized active node behavior still faces the mismatch problem; (b) Volume-preserved sampling corrects the mismatch problem.

We propose a solution that can guarantee the model generates the correct numbers of active nodes and edges for each timestep (See Fig. 2). This is achieved by simply reweighting the node and edge

Algorithm 1 Sovling $\alpha^*_{1:T}$ via binary search

Input: Loss and constraint tolerance ϵ_1, ϵ_2, $K_{\min}, K_{\max}$, degree sequence $\mathbf{d}^0$, and active node control $\gamma_{1:T}$.
$K_1 = K_{\min}, K_2 = K_{\max}, K = (K_1 + K_2)/2$
$\alpha^*_{1:T} = \arg\min_{\alpha_{1:T}} \mathcal{L}(\alpha_{1:T}; K, \gamma_{1:T}, \mathbf{d}^0)$
while $\mathcal{L}(\alpha^*_{1:T}; \cdot) > \epsilon_1$ or $\prod_{t=1}^T \alpha^*_t > \epsilon_2$ **do**
  if $\mathcal{L}(\alpha^*_{1:T}; \cdot) > \epsilon_1$ **then**
    $K_1 = K, K = (K + K_2)/2$
  else if $\prod_{t=1}^T \alpha^*_t > \epsilon_2$ **then**
    $K_2 = K, K = (K + K_1)/2$
  end if
  $\alpha^*_{1:T} = \arg\min_{\alpha_{1:T}} \mathcal{L}(\alpha_{1:T}; K, \gamma_{1:T}, \mathbf{d}^0)$
end while
Output: Edge noise schedule parameter $\alpha^*_{1:T}$

Algorithm 2 Volume-preserved sampling via reweighting

Input: Degree sequence $\mathbf{d}^0$, edge model p_θ, denoising timestep T, and edge noise schedule parameter $\alpha_{1:T}$.
Initialize $\mathbf{A}^T = \mathbf{0}$
for $t = T, \ldots, 1$ **do**
  Compute degree sequence $\mathbf{d}^t$ from $\mathbf{A}^t$
  Compute $\hat{q}(\mathbf{s}^t|\mathbf{d}^t, \mathbf{d}^0)$ using Eq. 5
  Sample $\mathbf{s}^t \sim \hat{q}(\mathbf{s}^t|\mathbf{d}^t, \mathbf{d}^0)$
  Compute $\hat{p}_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$ using Eq. 6
  Sample $\mathbf{A}^{t-1} \sim \hat{p}_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t, \mathbf{s}^t)$
end for
Output: Generated graph $\mathbf{A}^0$

distribution. For active node distribution, we have the following corrected form:

$$\hat{q}(\mathbf{s}^t|\mathbf{d}^t,\mathbf{d}^0)=\prod_{i=1}^N \hat{q}(\mathbf{s}_i^t|\mathbf{d}^t,\mathbf{d}^0),\ \hat{q}(\mathbf{s}_i^t|\mathbf{d}^t,\mathbf{d}^0)=\mathcal{B}\left(\mathbf{s}_i^t;\frac{h_{\mathbf{d}^0}(\alpha_{1:t},t)}{\sum_{i=1}^N p_i^t}p_i^t\right),\ p_i^t=1-\left(1-\frac{\beta_t\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\right)^{\mathbf{d}_i^0-\mathbf{d}_i^t}. \quad (5)$$

Recall that given edge noise schedule $\alpha_{1:T}$, $h_{\mathbf{d}^0}(\alpha_{1:t}, t)$ is the expected number of active nodes at timestep t. The reweighting is performed on active nodes that still have a degree budget, i.e., $\mathbf{d}_i^t < \mathbf{d}_i^0$. For edge distribution, we have

$$\hat{p}_\theta(\mathbf{A}^{t-1}|\mathbf{A}^t,\mathbf{s}^t,\mathbf{d}^0) = \prod_{i,j:i<j} \mathcal{B}\big(\mathbf{A}_{i,j}^{t-1};\frac{\Delta E_t}{\sum_{i',j':i'<j'}\ell_\theta^t(i',j')}\ell_\theta^t(i,j)\big), \quad (6)$$

$$\text{where } \ell_\theta^t(i,j) = \hat{p}_\theta(\mathbf{A}_{i,j}^{t-1}=1|\mathbf{A}^t,\mathbf{s}^t),\ \text{and } \Delta E_t = \big((\bar{\alpha}_{t-1}-\bar{\alpha}_t)\sum_{i=1}^N \mathbf{d}_i^0 + {\mathbf{s}^t}^T\mathbf{A}^t\mathbf{s}^t\big)/2.$$

Here $\ell_\theta^t(i,j)$ is the probability of forming an edge between node i and j. Note that we only compute probabilities for active node pairs, and $\hat{p}_\theta(\mathbf{A}_{i,j}^{t-1}=1|\mathbf{A}^t,\mathbf{s}^t) = \mathbf{A}_{i,j}^t$ if one of the $\{i,j\}$ is inactive. Note that in each denoising step, we regenerate all edges within the subgraph indicated by $\mathbf{s}^t$ as this allows the model to refine its previous prediction. The denominator of the weight in Eqn. 6 is the expected number of edges the model will generate within the subgraph. And the numerator ΔE_t represents the actual number of edges it should generate within the subgraph at that moment. With such reweighting, we can guarantee the model generates the correct number of edges at each timestep.

The correction operations on nodes and edges are optional and can be performed separately. We demonstrate the proposed sampling scheme in Alg. 2. With the corrected sampling algorithm, one can perform graph generation with edge overlap control [3].

4 Experiments

We demonstrate how the proposed techniques improve EDGE in terms of generative accuracy and efficiency. We denote the improved EDGE as EDGE++. We also present an application for generating realistic graphs with precise edge overlap control.

4.1 Setup

Datasets. We perform experiments on two large networks: Polblogs [1] and PPI [29]. The Polblogs network contains 1,222 nodes and 16,714 edges in total, and the PPI network contains 3,852 nodes and 37,841 edges.

Evaluation. We follow Chanpuriya et al. [3] and Chen et al. [7] to assess the consistency of the graph statistics between the generated networks and the original one. Our evaluation metrics encompass the following graph statistics: maximum degree; normalized triangle counts (NTC); normalized square counts (NSC); power-law exponent of the degree sequence (PLE); GINI; assortativity coefficient (AC) [23]; global clustering coefficient (CC) [3]; and characteristic path length (CPL). We also access the memory consumption of the models during training and sampling.

	Graph statistics								Memory usage (GB)	
	Max Deg.	NTC	NSC	PLE	GINI	AC	CC	CPL	Training	Sampling
					Polblogs					
True	351	1	1	1.414	0.622	-0.221	0.23	2.738	-	-
EDGE	**355.1**	**1.018**	1.052	**1.400**	**0.611**	-0.166	0.239	2.589	22.4	6.5
EDGE++	344.2	**1.016**	**1.023**	**1.401**	**0.603**	**-0.201**	**0.226**	**2.663**	**15.4**	**6.1**
					PPI					
True	593	1	1	1.462	0.629	-0.099	0.092	3.095	-	-
EDGE	**593.5**	1.143	1.601	**1.431**	**0.604**	-0.062	**0.102**	**3.071**	57.6	19.5
EDGE++	**594.1**	**0.905**	**1.254**	**1.440**	**0.612**	**-0.081**	**0.082**	3.011	**34.1**	**17.2**

Table 1: Generative performance of EDGE and EDGE++. For graph statistics, we report the average statistics over 8 generated samples, the numbers in bold indicate the method is better at the 5% significance level. Memory Usage of training is reported with a batch size of 4 and sampling with a batch size of 1.

Baselines. Since EDGE has shown superior results over traditional baselines [26; 27; 3], we only compare to EDGE in § 4.2. In § 4.3 We also compare against CELL [26], TSVD [27], and three methods proposed by Chanpuriya et al. [3] (CCOP, HDOP, Linear).

4.2 Generative Performance

We directly compare the generative performance of EDGE and EDGE++ in Table 1. EDGE++ achieves competitive or better performance than EDGE in terms of recovering the graph statistics. Specifically, EDGE++ excels in 7 out of 8 metrics in both Polblogs and PPI datasets. **We hypothesize the reason is that the complexity of the edge prediction task is amortized to each timestep, the model is then more capable of a relatively simpler learning task.** Moreover, training EDGE++ is more memory-economic compared to EDGE: it saves 31.25% and 40.78% of the GPU memory in training Polblogs and PPI datasets, respectively. This further demonstrates that with the proposed techniques, one can scale EDGE to model even larger graphs.

4.3 Graph Generation with EO Control – An Application

Chanpuriya et al. [3] shows that edge-independent graph models cannot generate desired graph statistics when the EO is low. Specifically, it tunes the EO to control the similarity between the generated graphs and the original graph. This section demonstrates how our proposed methods empower EDGE to generate graphs with controllable EOs. As we can observe in Fig. 3, the performance of EDGE degenerates when tunning EO from 0 to 1. However, such a phenomenon is not observed in EDGE++, indicating that we can synthesize realistic graphs with different levels of diversity. We further elaborate on how to control the EO of EDGE and EDGE++ in the App. B.2

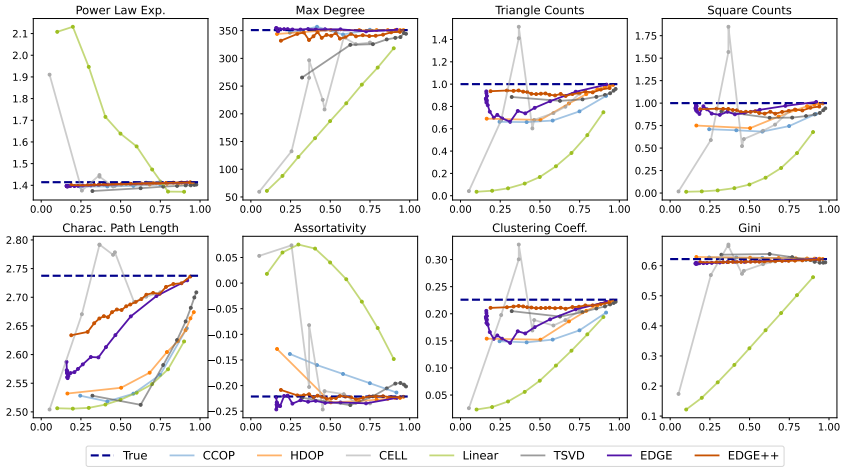


Figure 3. Graph generation of Polblogs with edge overlap control

5 Conclusion

In this work, we propose two techniques to improve the accuracy and efficiency of EDGE, a generative graph model that is able to generate high-quality large graphs. By customizing the number of active nodes at each timestep, EDGE requires significantly less GPU memory during training and achieves better generative performance on Polblogs and PPI databases. Moreover, the proposed volume-preserved sampling alleviates the trajectory mismatch problem, enabling one to generate graphs with edge overlap control. Our empirical study validates the effectiveness of the proposed techniques.

References

[1] L. A. Adamic and N. Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005.

[2] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann. Netgan: Generating graphs via random walks. In *International conference on machine learning*, pages 610–619. PMLR, 2018.

[3] S. Chanpuriya, C. Musco, K. Sotiropoulos, and C. Tsourakakis. On the power of edge independent graph models. *Advances in Neural Information Processing Systems*, 34:24418–24429, 2021.

[4] X. Chen, X. Han, J. Hu, F. J. Ruiz, and L. Liu. Order matters: Probabilistic modeling of node sequence for graph generation. *arXiv preprint arXiv:2106.06189*, 2021.

[5] X. Chen, X. Chen, and L. Liu. Interpretable node representation with attribute decoding. *arXiv preprint arXiv:2212.01682*, 2022.

[6] X. Chen, Y. Li, A. Zhang, and L.-p. Liu. Nvdiff: Graph generation through the diffusion of node vectors. *arXiv preprint arXiv:2211.10794*, 2022.

[7] X. Chen, J. He, X. Han, and L.-P. Liu. Efficient and degree-guided graph generation via discrete diffusion modeling. *arXiv preprint arXiv:2305.04111*, 2023.

[8] H. Dai, A. Nazi, Y. Li, B. Dai, and D. Schuurmans. Scalable deep generative modeling for sparse graphs. In *International conference on machine learning*, pages 2302–2312. PMLR, 2020.

[9] P. Erdos, A. Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[10] K. K. Haefeli, K. Martinkus, N. Perraudin, and R. Wattenhofer. Diffusion models for graphs benefit from discrete state spaces. *arXiv preprint arXiv:2210.01549*, 2022.

[11] X. Han, X. Chen, F. J. Ruiz, and L.-P. Liu. Fitting autoregressive graph generative models through maximum likelihood estimation. *Journal of Machine Learning Research*, 24(97):1–30, 2023.

[12] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[13] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[14] J. Jo, S. Lee, and S. J. Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. *arXiv preprint arXiv:2202.02514*, 2022.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[17] L. Kong, J. Cui, H. Sun, Y. Zhuang, B. A. Prakash, and C. Zhang. Autoregressive diffusion model for graph generation.

[18] J. Li, J. Yu, J. Li, H. Zhang, K. Zhao, Y. Rong, H. Cheng, and J. Huang. Dirichlet graph variational autoencoder. *Advances in Neural Information Processing Systems*, 33:5274–5283, 2020.

[19] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[20] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4255–4265, 2019.

[21] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] N. Mehta, L. C. Duke, and P. Rai. Stochastic blockmodels meet graph neural networks. In *International Conference on Machine Learning*, pages 4466–4474. PMLR, 2019.

[23] M. E. Newman. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002.

[24] M. E. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proceedings of the national academy of sciences*, 99(suppl_1):2566–2572, 2002.

[25] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 4474–4484. PMLR, 2020.

[26] L. Rendsburg, H. Heidrich, and U. Von Luxburg. Netgan without gan: From random walks to low-rank approximations. In *International Conference on Machine Learning*, pages 8073–8082. PMLR, 2020.

[27] C. Seshadhri, A. Sharma, A. Stolman, and A. Goel. The impossibility of low-rank representations for triangle-rich complex networks. *Proceedings of the National Academy of Sciences*, 117(11):5631–5637, 2020.

[28] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[29] C. Stark, B.-J. Breitkreutz, A. Chatr-Aryamontri, L. Boucher, R. Oughtred, M. S. Livstone, J. Nixon, K. Van Auken, X. Wang, X. Shi, et al. The biogrid interaction database: 2011 update. *Nucleic acids research*, 39(suppl_1):D698–D704, 2010.

[30] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

[31] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[32] C. Zang and F. Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 617–626, 2020.

A Derivation of Active Node Control

We are interested in computing the expected number of active nodes given a prescribed degree sequence and an edge noise schedule $\alpha_{1:T}$:

$$h_{\mathbf{d}^0}(\alpha_{1:t}, t) = \sum_{i=1}^{N} q(\mathbf{s}_i^t = 1 | \mathbf{d}_i^0), \tag{7}$$

where $q(\mathbf{s}_i^t | \mathbf{d}_i^0)$ is the distribution of node i being active at timestep t. Before deriving the form of the distributions, we first revisit the two properties derived by Chen et al. [7]:

Property 1. The forward degree distributions have the form

$$q(\mathbf{d}^{t-1}|\mathbf{d}^0) = \prod_{i=1}^{N} q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^0), \text{ where } q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^0) = \text{Binomial}(k = \mathbf{d}_i^{t-1}, n = \mathbf{d}_i^0, p = \bar{\alpha}_{t-1}). \tag{8}$$

Intuitively, for $q(\mathbf{d}^{t-1}|\mathbf{d}^0)$, there are $\mathbf{d}_i^0$ edges connected to node i, each with probability $\bar{\alpha}_{t-1}$ to be kept at time step $t-1$. The probability the number of remaining edges equals $\mathbf{d}_i^{t-1}$ at time step $t-1$ is a binomial distribution.

Property 2. At timestep t, the active node distribution for node i given $\mathbf{d}_i^{t-1}$ is

$$q(\mathbf{s}_i^t|\mathbf{d}_i^{t-1}) = \mathcal{B}\big(\mathbf{s}_i^t; 1-(1-\beta_t)^{\mathbf{d}_i^{t-1}}\big). \tag{9}$$

With the above properties, we show that when $t = 1$, we directly have

$$q(\mathbf{s}_i^1|\mathbf{d}_i^0) = \mathcal{B}(\mathbf{s}_i^1; 1-(1-\beta_t)^{\mathbf{d}_i^0}) = 1-\alpha_t^{\mathbf{d}_i^0} \tag{10}$$

by using property 1. For $t > 1$, $q(\mathbf{s}_i^t|\mathbf{d}_i^0)$ can be computed by marginalization, specifically, we introduce $\mathbf{d}_i^{t-1}$ and expand $q(\mathbf{s}_i^t|\mathbf{d}_i^0)$ into follow:

$$\begin{aligned}
q(\mathbf{s}_i^t|\mathbf{d}_i^0) &= \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} q(\mathbf{s}_i^t, \mathbf{d}_i^{t-1}|\mathbf{d}_i^0) \\
&= \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} q(\mathbf{s}_i^t|\mathbf{d}_i^{t-1}) q(\mathbf{d}_i^{t-1}|\mathbf{d}_i^0). \\
&= \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} \mathcal{B}(\mathbf{s}_i^t; 1-\alpha_t^{\mathbf{d}_i^{t-1}}) \text{Bin}(k = \mathbf{d}_i^{t-1}, n = \mathbf{d}_i^0, p = \bar{\alpha}_{t-1}) \\
&= \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} (1-\alpha_t^{\mathbf{d}_i^{t-1}}) \text{Bin}(k = \mathbf{d}_i^{t-1}, n = \mathbf{d}_i^0, p = \bar{\alpha}_{t-1}).
\end{aligned}$$

Now we show $q(\mathbf{s}_i^t|\mathbf{d}_i^0)$ as

$$q(\mathbf{s}_i^t = 1|\mathbf{d}_i^0) = \begin{cases} 1-\alpha_1^{\mathbf{d}_i^0}, & \text{if } t = 1 \\ \displaystyle\sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} (1-\alpha_t^{\mathbf{d}_i^{t-1}}) \text{ Bin}(k = \mathbf{d}_i^{t-1}, n = \mathbf{d}_i^0, p = \bar{\alpha}_{t-1}), & \text{otherwise} \end{cases}. \tag{11}$$

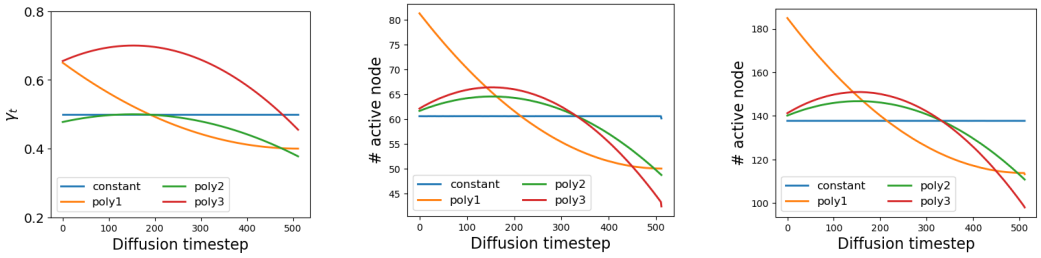
Then we can derive the expected number of active nodes as

$$h_{\mathbf{d}^0}(\alpha_{1:t}, t) = \begin{cases} \displaystyle\sum_{i=1}^{N} (1-\alpha_1^{\mathbf{d}_i^0}), & \text{if } t = 1 \\ \displaystyle\sum_{i=1}^{N} \sum_{\mathbf{d}_i^{t-1}=1}^{\mathbf{d}_i^0} (1-\alpha_t^{\mathbf{d}_i^{t-1}}) \text{ Bin}(k = \mathbf{d}_i^{t-1}, n = \mathbf{d}_i^0, p = \bar{\alpha}_{t-1}), & \text{otherwise} \end{cases}. \tag{12}$$

B Experiment Details

B.1 Experiment Setup

For both Polblogs and PPI datasets, we set the number for diffusion timesteps to 512. We use the same architecture from [7], with 5 message-passing blocks, each with 8 attention heads. We employ



(a) schedule of $\gamma_{1:T}$ (b) active node schedule in Polblogs (c) Active node schedule in PPI

Figure 4. Visualization of how different γ strategies may lead to different active node schedules on Polblogs and PPI datasets.

the Adam optimizer [15] with a weight decay of 10^{-4} and use a batch size of 4 for both datasets. The learning rate was fixed at 10^{-4}, and we did not employ any learning rate scheduler during the training process. For model evaluation, we selected the model that minimized the statistics difference between the generated graphs and the original graphs. We use a batch size of 8 during evaluation.

B.2 Graph Generation with EO control

We can control the EO between the generated graphs and the original graph by generating $\mathbf{A}^0$ from $\mathbf{A}^t$ with different $\mathbf{A}^t$. For any timestep $t \in [0, T]$, we can first draw $\mathbf{A}^t \sim q(\mathbf{A}^t|\mathbf{A}^0)$, then use the learned denoising model to draw $\mathbf{A}^0$ by sequentially drawing $(\mathbf{A}^{0:t-1}, \mathbf{s}^{1:t})$ from $p_\theta(\mathbf{A}^{0:t-1}, \mathbf{s}^{1:t}|\mathbf{A}^t)$. In the experiment of § 4.3, we choose $T = 512$ and time interval $\Delta = 25$, for each $t \in \{\Delta, 2\Delta, 3\Delta, \ldots, T\}$, we generate eight graphs and evaluate their statistics.

For better visualization, we choose constant active node scheduling for EDGE++, which leads to that the density of $\mathbf{A}^t$ decreases linearly as $t \to T$. Note that since EDGE doesn't support customization of the active node control, most of the sampled $\mathbf{A}^0$ appear to have a very low EO. Moreover, due to the mismatching issue, EDGE conversely performs worse in terms of recovering the true statistics as EO increases.

C Related Works

Edge-independent models, which assume the independent formation of edges with certain probabilities, are commonly found in probabilistic models for large networks. This category comprises various traditional models like the Erdős-Rényi graph models [9], SBMs [13], and deep neural models like variational graph auto-encoders [16; 22; 18; 5], NetGAN and its variant [2; 26]. Recent studies reveal that these models fail to replicate desired statistics of the target network, such as triangle counts, clustering coefficient, and square counts [27; 3].

On the other hand, deep auto-regressive (AR) graph models [19; 31; 20; 32; 11] construct graph edges by sequentially generating elements of an adjacency matrix. These algorithms are notably slow as they require making N^2 predictions. Dai et al. [8] propose a method to circumvent this by leveraging graph sparsity and predicting only non-zero entries in the adjacency matrix. However, these AR-based models often face long-term memory issues, making it difficult to model global graph properties. These models also lack invariance with respect to node orders of training graphs, necessitating specialized techniques for their training[4; 11].

Diffusion-based generative models have been demonstrated to be effective in producing high-quality graphs [25; 21; 14; 10; 6; 30; 17; 7]. They model edge correlations by refining a graph through multiple steps, overcoming the limitations of auto-regressive models. While the majority of the diffusion-based models have primarily focused on generation tasks with smaller graphs, EDGE [7] is the first model that scales to generate large graphs with thousands of nodes. However, EDGE is found to be inefficient in utilizing the denoising model.

D Additional Results

We investigate how to choose a suitable active node control for training EDGE. we found that constant schedule and polynomial schedule give better results in terms of generative performance. Specifically, we investigate the following three polynomial functions:

poly1: $\gamma_t = (0.5t/T - 0.5)^2 + 0.4$
poly2: $\gamma_t = (0.5t/T - 0.5)^2 + 0.5$
poly3: $\gamma_t = -0.5(t/T - 0.3)^2 + 0.7$

We visualize the chosen functions in Fig. 4(a). Moreover, we also demonstrate how the γ schedule maps to the actual node control in Polblogs (Fig. 4(b)) and PPI (Fig. 4(c)). The active node schedule is data-specific since it needs to make sure the constraint $\prod \alpha_t \approx 0$ is satisfied. This is achieved by tuning the parameter K as discussed in § 3.1.

We further provide the generative performance of EDGE++ using different active node schedules in Table. 2. All the active node schedules yield competitive performance in terms of recovering graph statistics. We report the result of the constant active node schedule in § 4.2.

	Max Deg.	NTC	NSC	PLE	GINI	AC	CC	CPL
				Polblogs				
True	351	1	1	1.414	0.622	-0.221	0.23	2.738
constant	344.2	1.016	1.023	1.401	0.603	-0.201	0.226	2.663
poly1	351.0	0.926	0.972	1.400	0.611	-0.171	0.205	2.634
poly2	352.3	0.922	0.913	1.400	0.613	-0.218	0.205	2.633
poly3	329.0	0.998	1.032	1.400	0.609	-0.173	0.225	2.673
				PPI				
True	593	1	1	1.462	0.629	-0.099	0.092	3.095
constant	594.1	0.905	1.252	1.440	0.612	-0.081	0.082	3.011
poly1	594.3	0.904	1.244	1.440	0.612	-0.081	0.088	3.029
poly2	584.5	0.765	0.966	1.440	0.611	-0.084	0.070	3.009
poly3	594.0	0.743	0.999	1.442	0.614	-0.099	0.059	3.026

Table 2: Generative performance